VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Webové aplikace využívající HTML 5

Web Applications Based on HTML 5

2012 Michal Daněk

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Michal Daněk**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Webové aplikace využívající HTML 5
Web Applications Based on HTML 5

Zásady pro vypracování:

Technologie webových aplikací je neustále se vyvíjející oblastí, kdy aktuálními tématy jsou např. HTML 5 a CSS 3. Cílem práce je zachycení možností této technologie, a to v kombinaci s technologiemi .NET MVC 3.

1. Popište aktuální nové vlastnosti a možnosti připravovaného standardu HTML 5 v kombinaci s CSS 3.
2. Specifikujte funkce a řešení, které jsou typické pro zvýšení interaktivity uživatelských rozhraní webových aplikací. Navrhněte jejich řešení pomocí přístupů HTML 5.
3. Implementujte komplexní ilustrační uživatelské prostředí podle předchozích závěrů. K implementaci využijte technologii .NET MVC 3.
4. Zhodnoťte navržené rozhraní na základě reálných uživatelských zkušeností a technických možností.

Seznam doporučené odborné literatury:

[1]Brian P. Hogan: HTML5 and CSS3: Develop with Tomorrow's Standards Today, Pragmatic Bookshelf, ISBN: 978-1934356685, 2011
[2] Jon Galloway a kolektiv: Professional ASP.NET MVC 3, Wrox, ISBN: 978-1118076583, 2011

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Michal Radecký**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: 15. 8. 2012

Podpis: Michal Daněk

Chtěl bych na tomto místě poděkovat zejména vedoucímu své bakalářské práce Ing. Michalu Radeckému, PhD. za jeho cenné rady a podněty.

Abstrakt

Cílem práce je shrnout a představit nová webová API souhrnně označovaná jako HTML5. Důraz je kladen především na kategorizaci těchto technologií a jejich základní popis a demonstraci. Dalším cílem je ukázat, jak tato API dokáží zvýšit interaktivitu systému pro uživatele a také zkvalitnit a zjednodušit proces vývoje. Nejpřínosnější části HTML5 obsažené ve webové ASP.NET MVC3/Razor aplikaci pak prakticky ukazují, jak tato API používat.

Klíčová slova: HTML5, CSS3, MVC3, ASP.NET, webová aplikace

The aim of this thesis is to summarize and present new web APIs collectively denoted as HTML5. The emphasis is mainly put on categorization of these technologies and on their description as well. Another aim is to show how these APIs can increase the user experience and how they can easy the development phase. The most valuable parts of the HTML5 standard are contained in our ASP.NET MVC3/Razor application which practicaly demonstrates how to use them.

Keywords: HTML5, CSS3, MVC3, ASP.NET, web application

Seznam použitých zkratek

Zkratka	Význam
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
MVC	Model–View–Controller
JSON	JavaScript Object Notation
API	Application Programming Interface
W3C	World Wide Web Consortium
WHATWG	Web Hypertext Application Technology Working Group
XML	Extensible Markup Language
BLOB	Binary Large Object
DOM	Document Object Model
SVG	Scalable Vector Graphics
RFC	Request for Comments
SSL	Secure Sockets Layer
DTD	Document Type Definition
DRY	Don’t Repeat Yourself

Seznam algoritmů

Obsah

Část I
Úvod

V této práci se zabývám přínosy zcela nových technologií souhrnně označovaných jako HTML5 a CSS3. Vývoj těchto standardů byl a ještě z části i je velice bouřlivý. Stojí za nimi dvě organizace: W3C (World Wide Web Consortium) a dále WHATWG (Web Hypertext Application Technology Working Group). Druhou jmenovanou tvořili původně zejména developeři společností Apple, Opera a Mozilla, kteří se od W3C oddělili v roce 2004 hlavně pro nesouhlas s přípravou standardu XHTML 2.0. Poněkud populárně lze v úvodu říci, že to byla doba, kterou lze nyní označit jako XML bublinu. Tedy snaha dostat XML jako značkovací jazyk pro popis všech možných typů dokumentů.

WHATWG se nesoustředili jen na očištění stávajících standardů, ale spíše se zamýšleli nad tím, jak je rozšířit a co aktuálně chybí webovým vývojářům k jejich efektivnější a pohodlnější práci. Finální podoba HTML5 vychází z W3C standardů, které ale čerpají právě ze standardů připravených konsorciem WHATWG. K definitivní změně kurzu došlo v roce 2007, kdy byla rozpuštěna pracovní skupina připravující XHTML 2 a byly vzaty ke standardizaci výsledky WHATWG. Více se již historií a politikou okolo vzniku HTML5 zabývat nebudeme a přejdeme postupně k obsahu jednotlivých vylepšení.

HTML5 je souborem nových standardů, které se drží následujících neformálních axiomů:

1. *"Don't Break the Web"* – co funguje, mělo by fungovat i nadále,
2. *"Pave the Cowpaths"* – developeři si našli obtížné cesty, jak některé problémy typicky řešit; pojďme tyto cesty a řešení standardizovat a zkulturnit,
3. *"Be practical"* – měli bychom řešit aktuální potřebu, požadavky a zejména ty nejpalčivější nedostatky.

Podpora HTML5 v prohlížečích je k dnešnímu dni různá. My budeme pro testování jednotlivých funkcionalit využívat prohlížeče následujících aktuálních verzí – viz tabulka1. Data o podílu jednotlivých desktopových prohlížečů na celosvětovém trhu jsou získána z [6].[1]

Zjednodušeně by se dalo říci, že podpora HTML5 a CSS3 je na výborné úrovni v prohlížečích založených na Webkitu (Chrome a Safari) a dále ve Firefoxu a Opeře. Internet Explorer 9 HTML5 téměř neimplementuje. To by se ale mělo změnit s příchodem Internet

[1] Pro zajímavost poznamenejme, že nejpoužívanějším webovým prohlížečem k datu 07/2012 je v České republice prohlížeč Chrome.

Prohlížeč	Aktuální verze	Engine	Podíl na trhu
Firefox	14.0.1	Gecko	19.69%
Chrome	21.0	Webkit	27.06%
Safari	5.1.7	Webkit	14.55%
Internet Explorer	9.0.8112	Trident	24.42%
Opera	12.00	Presto	4.73%

Tabulka 1: Podíl prohlížečů na trhu

Exploreru 10. Pro veškeré testování ukázek z této bakalářské práce tedy doporučujeme použít prohlížeč Chrome nebo Firefox. Kromě několika sémantických tagů implementují téměř vše. Na webu existuje velmi pěkná aplikace pro otestování podpory HTML5 (`http://html5test.com/`) v aktuálním prohlížeči. Výsledkem je vždy skóre prohlížeče udávající míru podpory HTML5 (čím více, tím lépe). Aktuální výsledky jsou:

1. Chrome: 422 bodů (vítěz)
2. Opera: 385 bodů
3. Firefox: 330 bodů
4. Safari: 278 bodů (nemá inline Audio/Video, nemá File API)
5. Internet Explorer: 138 bodů (nepodporuje podporu Forms – různé sémantické input tagy, nepodporuje práci s historií, nepodporuje File API, nepodporuje WebWorkers atd.)

Jak již bylo naznačeno, HTML5 je souborem různorodých specifikací a vylepšení, který je určen jak pro zvýšení interaktivity a zážitku z aplikace na uživatelské straně, tak i pro zkvalitnění a zjednodušení vývoje těchto aplikací. Mohu s jistotou říci, že příchodem HTML5 se web a webové aplikace posouvají o velký kus kupředu. V dalším textu se budu snažit zejména kategorizovat a popsat všechny nejpodstatnější rysy HTML5 a CSS3.

Výklad na některých místech může působit poněkud nesourodě. Přesně takové jsou ale normy HTML5 týkají se nejrozmanitějších částí fungování webu – od grafiky po výpočty na pozadí.

Kompletní znění norem W3C je možno nalézt zde:

`http://dev.w3.org/html5/spec/`

Verze WHATWG, která se nyní označuje jako tzv. “living standard”, je méně formální a více experimentálně (výzkumně) zaměřená. Tato verze je ke stažení zde:

`http://www.whatwg.org/html/`

Tyto specifikace byly základním použitým materiálem při tvorbě této práce.

CSS3

Nezávislým pokrokem kupředu je standard CSS3, který přináší mnoho užitečných prvků do CSS (Cascading Style Sheets) stylů. Jedná se o zpětně kompatibilní rozšíření standardu CSS 2.1. Mezi hlavní vylepšení patří například sofistikovanější podmínky pro výběr prvků na stránce nebo podpora animací, stínování, externích fontů, barevných přechodů a násobných pozadí atd.

Část II

Nové přínosy HTML5

HTML5 přichází s mnoha rozšířeními, z nichž ty nejdůležitější jednotlivě kategorizujeme a popíšeme v této kapitole. Pro shrnutí je uvádíme v tabulce 2. Kde to bude možné, uvedeme drobné ukázky relevantních částí kódu. Části kódu, které bezprostředně nesouvisejí s novými funkcemi, nebudeme pro přehlednost uvádět.

Název API / Rozšíření	**Popis**
WebSockets	Umožnění plně duplexní asynchronní komunikace mezi prohlížečem a serverem.
WebWorkers	Paralelní výpočty v JavaScriptu.
Data Storage	Key-value úložiště lokálních dat vázaných na doménu.
Application Cache	Selektivní cachování webových stránek, obrázků, skriptů...
History	Pokročilá práce s historií prohlížeče.
File API	Přímý přístup k souborovému systému hosta.
Drag & drop	Podpora pro drag & drop jednak v rámci prohlížeče, jednak drag & drop objektů pocházejících mimo prohlížeč.
Semantics	Nové sémantické elementy pro lepší identifikaci významů tagů.
Audio / video	Nativní podpora pro přehrávání vybraných formátů audia a videa.
Graphics	Podpora kreslení na Canvas a podpora vektorového jazyka SVG.

Tabulka 2: Vybrané funkcionality HTML5

Existují i další podstandardy, které se řadí k HTML5. My se ale budeme zabývat pouze těmi nejzásadnějšími a nejpřínosnějšími. Velice podrobný přehled o HTML5 lze najít v monografiích [5, 2] a dalších.

1 Základní principy fungování HTML5

Vylepšení HTML sahají od pokročilých nových funkcí až k úplně nejzákladnějším pilířům fungování webu. Standard HTML5 nyní po dlouhé době fungování webu definuje, jak

jednoznačně zkonstruovat DOM strom z nevalidní HTML stránky. Více si o tom řekneme v následující kapitole.

1.1 Robustní parsování

Specifikace HTML není krátkým čtením a to ani co do definice validního syntaxu HTML stránky. V daném množství webových stránek[2], které se vyskytují na dnešním internetu, je naprosté minimum těch, které jsou plně validní podle existujících norem. Jelikož chceme být při renderování HTML stránek na jedné straně poměrně tolerantní (nemůžeme odmítnout stovky milionů částečně nevalidních stránek) a na druhé straně ale konzistentní s normou, musejí se prohlížeče umět vypořádat s nevalidními stránkami nebo jejich částmi. Před příchodem HTML5 nebyl způsob, jakým opravovat chyby v dokumentech přesně specifikován. Mohlo se tak tedy stát, že jedna stránka vypadala různě v různých prohlížečích. HTML5 toto narovnává tak, že exaktně definuje algoritmus parsování HTML kódu i způsob, jak opravovat nevalidní stavy. Výsledkem je tedy na všech prohlížečích zcela identický DOM strom, a tím pádem i (téměř) jednotně vyrenderovaný výstup.

S příchodem HTML5 se sjednocuje jediný `DOCTYPE` v hlavičce HTML dokumentů a to:

```
<!DOCTYPE html>
```

Odpadává tedy nutnost odkazu a specifikace DTD schématu dokumentu.

2 Asynchronní komunikace client/server

Několik standardů v HTML5 se věnuje metodám, jak mezi sebou prohlížeč a server komunikují. Jedním z častých aspektů webových aplikací je real-time komunikace, a to na pozadí a bez nutnosti reloadovat stránku. Tento problém se dosud řešil různě, velkou nevýhodou však typicky byla vysoká režie trafficu takovýchto řešení. HTML5 ale ve svém WebSockets API přichází s API, které tento rutinní problém řeší snad nejlepším možným způsobem.

2.1 WebSockets

WebSockets, dále jen “sokety”, jsou bez nadsázky označovány jako jeden z největších kroků kupředu HTML5 vzhledem ke škálovatelnosti webových aplikací a client/server komunikaci

[2] Aktuálně se počet webových stránek na internetu odhaduje na desítky miliard. Přesné číslo určit víceméně nelze.

obecně. K dnešnímu dni, narozdíl od některých ostatních technologií balíku HTML5, jsou již plně standardizovány dokumentem RFC 6455.[3]

Velké procento webových aplikací ke svému provozu potřebuje dlouhotrvající obousměrnou komunikaci. Komunikace protokolem HTTP funguje z principu fungování tohoto protokolu v half-duplex režimu, kdy webový server výhradně odpovídá na dotazy klientské strany. Samotný webový server přitom nemůže komunikaci sám iniciovat. Pokud jsme chtěli na webové stránce zobrazovat nějaká pravidelně aktualizovaná data, např. aktuální kurzy měn nebo nově příchozí zprávy ze serveru, museli jsme před příchodem HTML5 použít klasické techniky jako je tzv. *polling*.

Pollingem označujeme algoritmus, kdy se klientská strana webové aplikace (tj. prohlížeč) periodicky dotazuje serveru, zda došlo k nějakým změnám. Velkou nevýhodou tohoto postupu je fakt, že data obvykle bývají aktualizována jen "jednou za čas", a dochází tak k mnoha a mnoha dotazům s odpovědí: "Nic nového." Každý takovýto dotaz a odpověď vyžaduje zaslání a příjem zprávy v celkové velikost řádově i v kilobajtech. Pokud si představíme vytížený webový server s mnoha uživateli (desítky až stovky tisíc), tímto postupem dochází k obrovskému plýtvání kapacitou přenosové linky i vytížením serveru.

HTML5 přichází s novou abstrakcí – *sokety*. Sokety simulují plný full-duplex režim, tedy komunikaci iniciovatelnou oběma stranami. Režie jednoho handshaku přitom klesá z kilobajtů na jednotky bajtů. Dramaticky se snižuje oproti pollingu i typická latence příjmu zprávy (cca 5x resp. až na úroveň pingu). Veškerá komunikace probíhá nad protokolem TCP přes standardní port 80 (HTTP). Sokety jsou tedy široce použitelné a nestrádají filtrováním komunikace na firewallech.

Popišme si nyní konkrétní použití této technologie na příkladu, jak navážeme spojení a jak přeneseme a zareagujeme na zprávu. V JavaScriptu v prohlížeči inicializujeme soketové spojení vytvořením instance třídy `WebSocket`:

```
var socket = new WebSocket('ws://myserver.com/service');
```

Všimněme si, že sokety mají své vlastní označení protokolu `"ws"`. Standard umožňuje i zabezpečenou komunikaci užitím SSL, tento protokol má pak označení `"wss"`. Po získání instance třídy `WebSocket` již můžeme komunikovat se serverem. K tomu slouží následující metody a callbacky:

```
socket.onmessage = function(e) {
  alert('Received from server: ' + e.data);
```

[3] Zkratka RFC znamená "Request for Comments". RFC dokumenty představují základní pilíře fungování Internetu.

```
}
socket.send('Hello server, how are you?');
```

a dále callbacky `onopen`, `onerror` a `onclose`.

- Metoda `send` odešle na server libovolný UTF-8 kódovaný řetězec.
- Událost `onopen` je vyvolána po navázání úspěšného (TCP) spojení se serverem.
- Událost `onmessage` je vyvolána po obdržení zprávy ze serveru. Handler události message je poté libovolná funkce `function(e) {...}` a v `e.data` je umístěna příchozí zpráva.
- Událost `onclose` je vyvolána v momentě, kdy je soket zavřen. K zavření soketu z klientské strany potom slouží metoda `disconnect`.

Celkově sokety představují opravdu zásadní sjednocení a zjednodušení dosud používaných technik pro periodickou synchronizaci mezi klientem a serverem. V oboustranném (full-duplexním) režimu můžeme posílat zprávy na server a asynchronně je přijímat. Režie tohoto protokolu je přitom dramaticky nízká a celkový typický traffic je snížen řádově 100x. Sokety jsou plně implementovány v aktuálních verzích prohlížečů Chrome, Firefox a Safari, dá se předpokládat, že podporu přinese i IE 10.

Jelikož ještě sokety nejsou implementovány ve všech desktopových prohlížečích a tím méně v prohlížečích mobilních, existují knihovny jako `socket.io`[4], které implementují mnoho metod pro polling serveru a použijí tu nejlepší dostupnou. Ideálně právě sokety, alternativně ale třeba Flash, AJAX long polling, forever iframe nebo JSONP polling. Popis těchto alternativních technik už přesahuje rámec této práce.

3 Ukládání dat, cachování

Ukládání dat, ať už trvalých nebo dočasných, zaznamenává veliký krok kupředu. Dosud používaná řešení pomocí *cookies* a to i pro situace, kde bychom raději použili něco jiného, dostávají s příchodem těchto API nové obrysy.

3.1 Data Storage

Webové aplikace mohou svá data ukládat v zásadě na dvou místech: na webovém serveru nebo na klientské stanici. Každý druh dat je typicky lépe ukládat na té či oné straně.

[4] `http://socket.io/`

Informace, jako je historie nákupů v internetovém obchodě nebo seznam příspěvků uživatele na blogovacím webu, je samozřejmě lepší uložit na serveru. Oproti tomu některé, zejména dočasné údaje uživatele (jako například preference skinu nebo podobně), je lépe uložit na klientské straně bez nutnosti je posílat stále tam a zpět.

Před HTML5 byly jedinou technologií umožňující ukládat data na klientské straně cookies. Cookies jsou využívány zejména pro identifikaci tzv. *session* prohlížeče, čímž umožníme nad bezestavovým protokolem HTTP udržovat stavy mezi jednotlivými dotazy. Nevýhodou cookies je mimo jiné poměrně nešikovný způsob, jakým je lze z webových stránek užít pomocí JavaScriptu, ale zejména fakt, že se při každém dotazu často odesílají tam a zpět, a tím zbytečně zatěžují přenosovou linku.

S příchodem HTML5 dostáváme do rukou data storage API, pomocí kterého můžeme na klientské straně ukládat téměř libovolná serializovatelná data. Způsob užití těchto dat je nyní zcela v režii programátora, data není nutno posílat na server, tak jako je tomu v případě cookies. Unikátním důsledkem existence ukládání dat na klientském počítači je možnost vytvořit tzv. *offline* aplikace, které používají právě lokální úložiště pro svoji práci. Offline aplikace si pak lze představit jednak permanentní (bez jakékoli interakce serveru), jednak řekněme "odpojené" (s dávkovou synchronizací odpojených dat v případě dostupnosti sítě).

Existují dva typy úložišť: *local storage* a *session storage*. Obě tato úložiště jsou vázána ne na konkrétní webovou stránku, ale globálněji na celou webovou doménu. Data tedy můžeme sdílet mezi jednotlivými stránkami typicky v rámci celé webové aplikace. Local storage je určeno k ukládání permanentních dat, která zůstanou uložena i po restartu prohlížeče, zatímco session storage, jak název napovídá, použijeme pro uložení dočasných dat v rámci jedné session. Session končí zavřením záložky (tabu) prohlížeče. API pro zápis a čtení dat z obou úložišť připomíná key–value slovník.

Zápis do úložiště provedeme následovně:

```
localStorage['pocet'] = 123;
sessionStorage['visited'] = new Date();  /* Aktualni cas */
```

Naopak čtení:

```
alert(localStorage['pocet']);

/* Test na neexistenci klice */
if (localStorage['pocet'] == null) { ... }
```

Každé úložiště typicky umožňuje záznamy zapisovat, číst a také mazat. K tomu jsou určeny metody:

- `localStorage.clear()` pro smazání všech záznamů,
- `localStorage.removeItem('klic')` pro smazání právě jednoho vybraného záznamu.

Funkce data storage už tak, jak jsme je popsali dosud, obohacují webové aplikace zásadním způsobem. HTML5 zavádí navíc ještě možnost registrace callbacku `onstorage`, který je vyvolán v případě jakékoliv změny v úložištích. Registrovat jej můžeme následujícím standardním způsobem:

```
window.addEventListener('storage', my_callback, false);
```

Callback `my_callback` pak má následující strukturu:

```
function my_callback(e) {
/*
 * e.key - klíč
 * e.oldValue - původní hodnota
 * e.newValue - nová hodnota
 * e.url - doména
 */
 ...
}
```

Kapitolu uzavřeme jednoduchým příkladem, jak počítat počet navštívení dané stránky pomocí `localStorage` (tj. na klientské straně) – viz Algoritmus 1.

3.1.1 Je nutno být online

V souvislosti s data storage, tak jak jsme je popsali výše, je nutno zmínit jednu nečekanou zkušenost: "Je nutno být *online*!" V některých prohlížečích je data storage k dispozici, právě když je webová stránka dostupná online. A to ať už po internetu, anebo i přes lokální server. Stránka ale *nesmí* být otevřena přes `file://`, nýbrž plnohodnotným http(s) přístupem. Nejlépe na tom je Chrome, který v offline módu pouze zakáže odchytávat některé události. Horší je již podpora ve Firefoxu, který vytváří dojem, že vše je dostupné, veškeré uložené hodnoty jsou ale bez upozornění zahozeny. Internet Explorer v offline módu nenabízí vůbec nic.

Ukázka 1 Počítadlo navštívení stránek pomoc `localStorage`

```
<!doctype html>
<html>
  <head>
    <script>
    if (localStorage['pocet'] == null) {
      /* záznam je prázdný, jsme zde poprvé */
      var n = localStorage['pocet'] = 1;
    } else {
      var n = Number(localStorage['pocet']);
      n += 1; localStorage['pocet'] = n;
    }
    alert('Pocet otevreni stranky: ' + n);
    </script>
  </head>
  <body></body>
</html>
```

✎ Data storage je možno použít, právě když je webová stránka nahrána *online*. V offline módu tento systém téměř nefunguje.

3.1.2 Složitější objekty

Dosud jsme si ukázali, jak ukládat a číst základní datové typy. Relevantní požadavek samozřejmě je, abychom si mohli ukládat i libovolné naše objekty. K tomu slouží metody rozhraní JSON (JavaScript Object Notation). Pro detailní popis jazyka JSON viz kniha [8]. JSON není nic jiného než textový způsob, jak zapsat stromová data. Odhlédneme-li od rozšíření XML jako namespace a podobně, je JSON de facto totéž, jen o dost kompaktnější.

Každopádně rozhraní `JSON` nabízí funkci `JSON.stringify(obj)`, která serializuje objekt do jeho textové JSON reprezentace. V opačném směru pak existuje funkce `JSON.parse(json_obj)`, která řetězec převádí zase nazpět.

✎ Pro ukládání složitějších objektů je nutno použít serializaci přes JSON.

Ukázka 2 Příklad manifest souboru `manifest.mf`

```
CACHE MANIFEST
# 7.8.2012 15:30

CACHE:
index.html
css/base.css
img/logo.jpg

NETWORK:
compute.php

FALLBACK:
*.html /offline-banner.html
```

3.2 Application Cache

Cílem rozšíření HTML5 jménem Application Cache (dále jen Cache) je umožnit webovým aplikacím částečně fungovat i v případě nedostupnosti serveru. Současné prohlížeče samozřejmě dávno obsahují cache, nicméně z pohledu programátora typicky buď nejsou přístupné, nebo nemají žádné jednotné API.

Základem pro práci s Cache je textový manifest soubor. Uveďme si jednoduchý příklad takovéhoto souboru `mymanifest.mf`.

Název tohoto souboru je volitelný a užití konkrétního manifestu pro konkrétní stránku deklarujeme přímo v hlavičce HTML stránky v kořenovém elementu `<html>`. Můžeme použít jak lokální název souboru, tak i obecné URL.

```
<html manifest="mymanifest.mf">
...
</html>
```

Prohlížeč při požadavku na danou stránku nahraje odpovídající manifest a cachuje soubory v něm uvedené podle syntaxe, kterou rozvedeme dále. Zásadní přitom je, že cachované verze souborů se užívají tak dlouho, dokud nedojde k aktualizaci manifestu. Tedy jinými slovy, pokud aktualizujeme např. obrázek `img/logo.jpg`, ale manifest ponecháme beze změny, bude prohlížeč stále zobrazovat původní verzi loga. K nahrání nové verze jej přinutíme až aktualizací manifestu.

Takto cachované zdroje jsou dostupné i bez přístupu na internet. Application Cache funguje k dnešnímu stavu bezchybně v prohlížeči Firefox 14.

3.2.1 Struktura manifestu

Struktura manifestu je poměrně jednoduchá:

- První řádek obsahuje pevně text `CACHE MANIFEST`, dále se soubor skládá z několika sekcí `CACHE`, `NETWORK`, `FALLBACK`.
- Sekce `CACHE:` Jedná se o implicitní sekci. Pokud neuvedeme v manifestu žádný název sekce, použije se právě tato. Soubory uvedené v této sekci jsou cachovány a přístupny i v **offline režimu**.
- Sekce `NETWORK:` Dotazy na soubory (zdroje) uvedené v této sekci mají zakázáno použít cache, a jsou tedy dostupné pouze v **online režimu**.
- Sekce `FALLBACK:` Sekce `FALLBACK` je velmi užitečná. Definuje, které stránky se mají ke kterým zdrojům zobrazit jako alternativa v případě, že jsou původní zdroje nedostupné. Můžeme tak například zobrazit jednotnou stránku `offline-banner.html` v případě, že se uživatel dotazuje na některou jinou stránku v offline režimu.

3.3 Práce s historií prohlížeče

V poslední dekádě zaznamenal web vznik webových aplikací. Předtím web sloužil víceméně jako úložiště statických webových stránek propojených odkazy. Původní URL byla typicky ve tvaru `http://domain.com/page.html`, dotazem na takováto URL uživatel získal HTML stránku, a tím vše skončilo. Dnes u webových aplikací ale chceme na jedné stránce (spíše URL) zachytit mnoho rozličných stavů aplikace a zejména chceme, aby bylo možno si pro různé stavy vytvořit bookmarky a následně se k nim vracet. K uchování stavů využijeme pomocí HTML5 jednak hash-část URL (viz níže), jednak perzistentní úložiště objektů asociovaných s různými URL. Toto je výhodné zejména u aplikací se silným využitím JavaScriptu na pozadí, kdy se stavy aplikace mění bez nutnosti znovu načíst stránku.

3.3.1 Hashchange

URL v prohlížeči se skládá z několika částí. Hash je ta část URL, která následuje za symbolem # a to pro naše případy včetně tohoto znaku. Změny v této části URL můžeme odchytit handlerem události takto:

```
window.onhashchange = function() {
  alert('We detected change in hash.');
}
```

Všechny aktuální verze prohlížečů tuto událost podporují. Zachycení těchto změn je klíčové pro vývoj rychlých a uživatelsky komfortních webových aplikací. To si osvětlíme v následujícím odstavci.

3.3.2 Historie

Při práci s prohlížečem, v tomto kontextu typicky v rámci práce s webovou aplikací a ne náhodném brouzdání, navštěvujeme mnoho různých stránek (URL). Prohlížeče obsahují API jak přistupovat a manipulovat s takto vzniklou historií URL, a to pomocí objektu `window.history`:

- `window.history.back()` – jde jeden krok vzad,
- `window.history.forward()` – jde jeden krok vpřed,
- `window.history.go(nr_steps)` – jde o `nr_steps` kroků vpřed (akceptuje i záporné hodnoty parametru `nr_steps`),
- `window.history.length` – délka historie.

HTML5 zavádí nové metody `pushState()` a `replaceState()` objektu `window.history`. Rozeberme si je postupně:

- `pushState(obj, title, url)` – tato metoda uloží do historie nový záznam a tento záznam se stane aktuálním. Parametr `obj` jsou libovolná serializovatelná data. Můžeme si tak uložit různé užitečné informace nutné k rekonstrukci daného stavu. Pokud uvedeme parametr `url`, prohlížeč v případě použití tohoto záznamu nastaví dané URL, a to bez reloadu stránky. Nové URL musí mít stejnou doménovou část, můžeme ale upravovat libovolně path a hash část URL. Je důležité zmínit, že změna URL tímto způsobem nevyvolá událost `hashchange`.
- `replaceState(obj, title, url)` – chová se stejně jako `pushState` s jedinou změnou, že současný stav je nahrazen (nelze tedy v prohlížeči stisknout tlačítko Zpět).

Dále máme k dispozici událost `window.onpopstate`, která je vyvolána kdykoliv při stisku tlačítek Zpět nebo Vpřed (nebo odpovídajících metod). Veškeré stavy `obj`, tak jak je výše uvedenými metodami přidáváme na zásobník, se ukládají na *disku*. Tyto stavy jsou pak předány handleru události `onpopstate`.

4 Přímý přístup k souborovému systému

Webové aplikace na své klientské JavaScriptové straně dosud neměly žádný přímý způsob, jak číst lokální soubory. Proto například pro sofistikovanější nahrávání více souborů na server najednou byly nezřídka využívány technologie jako Flash, které tento přístup měly díky tomu, že se jednalo o samostatné pluginy. File API zavádí krom jiného zejména třídu `FileReader`, pomocí níž je možno asynchronně načítat lokální soubory včetně podpory načítání souborů po blocích. Toho využíváme v našem demu pro multiupload velikých obrázků.

4.1 File API

File API je nyní ve stádiu W3C Working Draft (12 July 2012) a umožňuje přímý přístup k souborovému systému klientské stanice. API sestává z několik základních tříd, zejména:

- `File` – readonly soubor včetně doprovodných metadat jako název souboru, velikost, mime atp.,
- `FileList` – seznam souborů; využívá se např. při přetažení několika souborů pomocí drag & drop do prohlížeče,
- `Blob` – úsek binárních dat, využíváme např. u segmentovaného uploadu souboru, kdy uploadujeme soubor po 1 MB velikých úsecích,
- `FileReader` – třída pro čtení obsahu souborů; čtení se děje asynchronně, po přečtení požadovaných dat je vyvolána událost `FileReader.onload`.

4.1.1 Čtení ze souboru

Pro čtení ze souboru slouží několik metod třídy `FileReader`:

1. `FileReader.readAsBinaryString(Blob` nebo `File)`
2. `FileReader.readAsText(Blob` nebo `File, optional_encoding)`
3. `FileReader.readAsDataURL(Blob` nebo `File)`
4. `FileReader.readAsArrayBuffer(Blob` nebo `File)`

Průběh a stavy při čtení souboru lze monitorovat pomocí událostí:

- `onloadstart`, `onprogress`, `onload`,

Ukázka 3 Čtení souborů přes `FileReader` po vícenásobném výběru

```
<input id="files" type="file" name="files[]" multiple />
<output id="dst"></output>
<script type="text/javascript">
  function afterFilesPicked(evt) {
    var files = evt.target.files; /* Získali jsme FileList */
    var dst = [];
    var reader = new FileReader();
    reader.onload = function(e) { ... }; /* Nějaká akce */
    for (var i = 0, f; f = files[i]; i++) {
      reader.readAsDataURL(f);
      dst.push('<li>', escape(f.name), '</li>');
    }
    $('#dst').html('<ul>' + dst.join('') + '</ul>');
  }
  $('#files').change(afterFilesPicked);
</script>
```

- `onerror`, `onabort`,
- `onloadend`.

Uveďme si ještě kompletní příklad – viz Algoritmus 3. Vyrenderujeme tag pro multi-select souborů na disku. Po výběru těchto souborů se dynamicky vyrenderuje seznam jejich názvů.

4.1.2 Zápis do souboru

Zápis do souboru je nyní plně implementován v prohlížeči Chrome. Uveďme si ukázku, jak zapsat z JavaScriptu do dočasného souboru. Přístupný není plný souborový systém, ale tzv. *sandbox*, tj. zcela oddělený souborový prostor. Pro volbu dočasného souboru slouží flag `window.TEMPORARY`, pro trvalé zápisy pak `window.PERSISTENT`.

Použijeme k tomu metodu `window.requestFileSystem`, Chrome obsahuje rovněž alias `window.webkitRequestFileSystem`. Příklad je uveden v Algoritmu 4.

Popišme krátce fungování tohoto příkladu:

1. Zavoláme metodu `requestFileSystem` s požadavkem na dočasný souborový systém dané velikosti a uvedeme dvě metody: `oninitfs` a `onerror`.
2. Získáme `DirectoryEntry` objekt `root`.

Ukázka 4 Zápis do souboru File API

```
function oninitfs(fs) {
  var root = fs.root; /* DirectoryEntry objekt */
  root.getFile('data.txt', {create: true}, function(f) {
    f.createWriter(function(writer) {
      writer.onwrite = function(e) { ... }
      writer.onerror = function(e) { ... }
      var builder = new BlobBuilder();
      builder.append('My data');
      writer.write(builder.getBlob('text/plain'));
    }, onerror);
  }, onerror);
}
window.requestFileSystem(window.TEMPORARY, 1024*1024,
  oninitfs, onerror);
```

Ukázka 5 Detekce přítomnosti File API

```
<script>
if (window.FileReader && window.File
    && window.FileList && window.Blob) {
  alert('File API v plné síle!');
} else {
  alert('File API není plně podporováno!');
}
</script>
```

3. Metodou `getFile` zaregistrujeme callback na práci se souborem `data.txt`, soubor vytvoříme, pokud neexistoval.

4. Pomocí metody `createWriter` získáme `FileWriter` objekt a zapíšeme binární data vytvořená a uložená třídou `BlobBuilder`.

5. Jelikož práce se souborovým systémem může kdykoliv selhat, je důležité na všech odpovídajících místech registrovat obsluhu události `error`.

4.1.3 Detekce přítomnosti File API

Jelikož se jedná nyní ještě o Working Draft, je nutné pro zajištění portability mezi prohlížeči za běhu detekovat, zda je File API použitelné. To můžeme jednoduše udělat způsobem uvedeným v Algoritmu 5.

5 Interaktivita

V sekci týkající se zvýšení interaktivity s uživatelem se budeme věnovat zejména lepší podpoře pro drag & drop. Přetahovat můžeme jakékoliv DOM objekty pouze označením příslušného atributu, novinkou je pak dále možnost drag & drop mezi prohlížečem a jinými aplikacemi (např. Průzkumník).

5.1 Drag and drop

Drag and Drop je nové API umožňující uživateli jednoduchým způsobem přemisťovat objekty, a to jak v rámci stránky, tak i mimo ni. Požadovaný objekt stačí "uchopit" myší, "táhnout" nad požadované místo a "upustit". Rozlišujeme tři typy drag & drop situací:

1. Nativní drag & drop – tažení DOM objektů v rámci HTML stránky,
2. Drag-in – tažení objektů z plochy (nebo odjinud mimo stránku) do HTML stránky,
3. Drag-out – tažení objektů z HTML stránky ven na plochu (resp. do Průzkumníka nebo Nautila apod.).

HTML5 podporuje všechny tyto typy.

5.1.1 Nativní drag & drop

Abychom mohli takovouto funkcionalitu implementovat, využíváme některých nových DOM událostí, HTML5 atributů a JavaSript metod. V první řadě (v případě DOM objektů) musíme definovat, který z objektů bude určen k přetahování. To zajistíme pomocí atributu `draggable = true`. Dále je zapotřebí definovat prostor, do kterého bude možno přetahované elementy umisťovat. K tomu nám slouží atribut `dropzone = true`.

Jakmile uživatel uchopí objekt, je vyvolána DOM událost `dragstart`, kterou zachytáváme a obsluhujeme JavaScriptem. Název této metody specifikujeme v atributu `ondragstart` přetahovaného elementu. Druh přemisťovaných dat nastavíme JavaScript metodou `setData`. Když se přetahovaný objekt nachází nad prostorem, kde je jej možno upustit, je vyvolána událost `dragover`. I v tomto případě nastavujeme jméno JavaScript metody, pomocí které tuto událost obsluhujeme a to atributem `ondragover` elementu. V případě, že nyní uživatel objekt upustí, vznikne událost `drop`. Atributem `ondrop` opět definujeme název metody, pomocí které bude tato událost obsloužena. Tento postup demonstruje následující příklad 6. Použijeme `<div>` a nastavíme mu styl.

Ukázka 6 Ukázka struktury drag & drop

```
<style type="text/css">
  #dropArea {
    width: 400px;
    height: 50px;
    background-color: yellow;
  }
</style>
<div id="dropArea"
         dropzone="move␣s:text/plain"
         ondrop="dropHandler(e)"
         ondragover="overHandler(e)">
Drag items here</div>
<ul ondragstart="dragStart(e)">
        <li draggable="true" data-value="1">Prvni</li>
        <li draggable="true" data-value="2">Druhy</li>
        <li draggable="true" data-value="3">Treti</li>
</ul>
```

Celkový příklad použití drag & drop je nejlépe vidět na uploadu obrázků do obrázkového boxu v přiložené webové aplikaci.

6 Nové strukturální a sémantické elementy

Většina dnešních webových stránek používá pro vytváření své struktury především elementy `<div>`. Jaký obsah se v daném div elementu nachází, je pak upřesněno identifikátorem. Zdrojový kód pak obsahuje části podobné těm v ukázce 7.

Problém takto implementované struktury je především v její strojové nečitelnosti. Element `<div>` je příliš sémanticky obecný a neříká tak nic o svém obsahu. Tento problém řeší HTML5 svými novými strukturálními a sémantickými tagy. Programátor tak má možnost logicky, přehledně a čitelně strukturovat svůj dokument. Uveďme si nyní tyto nové sémantické tagy v tabulce 3.

7 Grafika a multimedia

HTML5 přichází se dvěma API, která se týkají kreslení. Prvním je Canvas API a druhým pak nativní podpora Scalable Vector Graphics (formát SVG). Canvas API je flexibilní JavaScriptové API, které nám umožňuje v prohlížeči kreslit. Pomocí elementu `<canvas>`

Tag	Použití
`header`	Vyznačuje hlavičku nebo její části.
`footer`	Vkládá zápatí nebo jeho části.
`hgroup`	Element hgroup používáme tehdy, chceme-li na stránku umístit víceúrovňové nadpisy. Tyto nadpisy pak uzavíráme do značek `<h1>` až `<h6>` a elementem `<hgroup>` je sjednocujeme.
`article`	Tento element používáme k vytvoření nezávislého obsahového celku, který má vlastní hlavičku, obsahovou část a patičku. Může reprezentovat například jeden článek na zpravodajském webu, jeden komentář apod.
`section`	Slouží k sjednocení určitých souvisejících částí webové stránky. Je podobný elementu `<article>`, narozdíl od něj je obsahově závislý (odkazující se) na jiné části dokumentu.
`aside`	Používáme k vymezení části textu. Můžeme jej použít například k oddělení odkazů, vysvětlivek, citací nebo části textu, který má přitáhnout čtenářovu pozornost.
`nav`	Vymezuje prostor pro navigační odkazy.
`bdi`	Tento element nám umožňuje obousměrně izolovat část textu, která pak může být kódována jinak než text okolní. Používáme jej v případě, kdy je do stránky klientem vkládán text v předem neznámém kódování.
`details`	Ohraničuje dodatečné podrobné informace na stránce. Uživatel má navíc možnost zobrazit nebo skrýt tyto informace. Toho docílí kliknutím na text, který uzavřeme značkami `<summary>`.
`summary`	Vymezuje viditelné části detailu stránky, jak je popsáno výše.
`figure`	Element sloužící k umisťování objektů jako jsou například obrázky, grafy, diagramy apod. Výhodou je možnost přidat k takovému objektu i popis či titulek, pomocí podřízeného elementu `<figcaption>`. Takto lze vytvořit např. ucelený objekt obsahující obrázek s popisem bez nutnosti používat element `<div>`.
`mark`	Slouží k vyznačení částí textu, které chceme na stránce graficky zvýraznit (typicky barevným podbarvením). Používáme nejčastěji k vyznačování hledaných slov na stránce.
`wbr`	Vyskytují-li se na naší stránce delší slova, u kterých chceme dovolit jejich případné rozdělení, pak poslouží tento element. Pomocí něj můžeme ve slovech vyznačit ta místa, která jsou pro rozdělení vhodná (povolená).
`time`	Vložení data, času nebo jejich kombinace nám umožňuje element `<time>`. Takto vložený datum a čas je pak viditelný pro různé druhy algoritmů strojových vyhledávání apod.

Tabulka 3: Nové strukturální elementy HTML5

Ukázka 7 Klasické dělení stránky pomocí `divů`

```
<div id="header">
  <h1>Zpravodajstvi - nove</h1>
</div>
<div id="sidebar">
  <ul>
    <li><a href="eko.html">Ekonomika</a></li>
    <li><a href="fin.html">Finance</a></li>
   <li><a href="kul.html">Kultura</a></li>
  </ul>
</div>
<div class="post">
  <h2>Zlodej zatcen</h2>
  <p>Policiste zatkli zlodeje, ktery minuly tyden..</p>
</div>
<div class="post">
  <h2>Koruna klesa</h2>
  <p>Kurz koruny vuci euru tento tyden....</p>
</div>
<div id="footer">
  <p>Copyright 2000-2012 Michal s.r.o.</p>
  <p>All Rights Reserved</p>
</div>
```

na HTML stránce získáme tzv. *canvas context*, pomocí kterého pak můžeme volat různé kreslicí operace. K dispozici máme (resp. budeme mít) dva základní módy 2D a 3D. 3D mód (tzv. WebGL) ještě není k dnešnímu dni běžně podporován, budeme se nyní spíše věnovat 2D režimu, jehož použití je rovněž demonstrováno i v naší aplikaci pro kreslení náhledů stránek.

7.1 Canvas 2D API

Canvas[5] ([7]) je nový HTML5 element, který nám přináší API pro tvorbu 2D grafiky. Jedná se o plátno, které umístíme na stránku a pomocí JavaSript metod na něj kreslíme. Takto můžeme vytvářet a za běhu modifikovat různé 2D objekty, jako jsou linky, křivky ale i různé barevné přechody, transformace, obrázky apod. Je možné pracovat také s videem, a to na úrovni jednotlivých pixelů. Jak s elementem canvas pracovat, si ukážeme na následujícím příkladu.

Naše plátno definujeme pomocí značky `<canvas>` a atributů určujících jeho identifikátor

[5] Angl. plátno

Ukázka 8 Užití nových sémantických `divů`

```
<header>
  <h1>Zpravodajstvi - nove</h1>
</header>
<nav>
  <ul>
    <li><a href="eko.html">Ekonomika</a></li>
    <li><a href="fin.html">Finance</a></li>
    <li><a href="kul.html">Kultura</a></li>
  </ul>
</nav>
<article>
  <h2>Zlodej zatcen</h2>
  <p>Policiste dnes zatkli zlodeje, ktery minuly...</p>
</article>
<article>
  <h2>Koruna klesa</h2>
  <p>Kurz koruny vuci euru tento tyden...</p>
</article>
<footer>
  <details>
    <summary>Copyright</summary>
    <p>Copyright 2000-2012 Michal s.r.o.</p>
    <p>All Rights Reserved</p>
  </details>
</footer>
```

a rozměry. Text uvnitř elementu bude zobrazen v případě, kdy prohlížeč nerozpozná značky `<canvas>`.

Ukázka 9 Vložení `canvas` elementu do stránky

```
<canvas id="d1" width="300" height="300">
        Váš prohlížeč nepodporuje Canvas.
</canvas>
```

Dále uložíme do proměnné `canvas` náš canvas objekt, který získáme na základě jeho identifikátoru metodou `getElementById()`. Na tomto objektu zavoláme funkci `getContext()` s parametrem `'2d'`, neboť chceme kreslit dvourozměrné objekty. Tato metoda nám vrátí `CanvasRenderingContext2d ctx;` rozhraní, na kterém budeme spouštět další kreslicí funkce.

Ukázka 10 Použití canvasu

```
<script type="text/javascript">
  var ctx = document.getElementById('d1').getContext('2d');
  ctx.fillStyle = 'blue';
  ctx.fillRect(0, 0, 20, 20);
  ctx.strokeStyle = '#f0f0f0';
  ctx.lineWidth = 3;
  ctx.lineCap = 'round';
  ctx.arc(100, 100, 20, 0, Math.PI, false);
  ctx.stroke();
</script>
```

Nyní již můžeme začít kreslit pomocí volání metod a definování proměnných na objektu `ctx`. Canvas API je standardní API analogické jiným grafickým systémům. Canvas API nepoužívá žádný souborový formát a kromě řídicího elementu nevyžaduje žádný další. Vše, co máme k dispozici, je čistá bitmapa, na kterou lze kreslit. Definovány jsou zejména následující funkce:

- kreslení obrazců,
- vyplňování barvami,
- vytváření gradientů a vzorků,
- kreslení textu,
- kopírování obrázků do canvas, videa a kopírování mezi canvasy navzájem,
- práce s jednotlivými pixely,
- exportování aktuálního canvasu do souboru,
- ukládání aktuálního stavu canvasu na zásobník a vracení se k předchozímu stavu (`save()`, `restore()`)

Kreslení na canvas je analogické podobným *bitmapovým* API z jiných programovacích jazyků. Mimo jiné Canvas API nemá žádnou podporu pro vrstvy, jak je známe například z Photoshopu. Aktuální verze prohlížečů již všechny podporují Canvas API, a to dokonce včetně hardwarové akcelerace, pokud je na konkrétním systému k dispozici.

7.2 SVG

SVG je jazyk pro definici vektorové grafiky pro web založený na XML. Jeho výhody a nevýhody oproti Canvas API výplývají z obecných výhod/nevýhod vektorové grafiky oproti bitmapové. Objekty je možné zvětšovat/zmenšovat bez ztráty kvality, jednoduše je editovat v jakémkoliv textovém editoru, komprimovat apod. Všechny SVG objekty lze navíc jednoduše animovat, což vidíme jako velkou výhodu ve srovnání s objekty Canvas, kde dosáhneme efektu animace pouze jejich neustálým překreslováním. Na druhou stranu SVG formát neumožňuje editovat video, kreslit trojrozměrnou grafiku, není vhodný např. na fotografie apod. SVG je tedy ideálním nástrojem pro generované obrázky, diagramy, grafy apod. Objekty mohou mít podobu linek, křivek, obdelníků, kruhů, elips, ale i oblouků, kubických a kvadratických Bézierových křivek apod. Vkládat můžeme také text a obrázky. Na objekty můžeme aplikovat barevné přechody, transformace, animace a také nastavovat jejich vzhled pomocí kaskádových stylů, což speciálně je velice užitečné.

Oblast pro kreslení SVG definujeme pomocí HTML značky `<svg>`, ke které doplníme atribut obsahující namespace. U zanořených svg značek je možné uvést i atribbuty pro definici polohy vůči nadřízenému elementu pomocí souřadnic.

Ukázka 11 Vložení `svg` elementu do stránky

```
<svg xmlns="http://www.w3.org/2000/svg">
...
  <circle id='x' cx='' cy='' r='' fill='blue' />
...
</svg>
```

Jak již bylo řečeno, SVG obrázek de facto programujeme pomocí XML elementů. Jelikož jednotlivé grafické objekty existují v DOM, je možno na nich odchytávat události tak jako na kterémkoli jiném elementu DOM stromu.

7.2.1 Srovnání Canvas API a SVG

Je důležité srovnat Canvas API s formátem SVG. Obě technologie řeší určitý problém a je velmi dobré vědět, kdy kterou zvolit. SVG podporuje vektorové kreslení, má podporu pro vrstvy. SVG je součástí DOM, a je tak možno jednoduše nastavovat jednotlivým částem obrázku různé události. Je rovněž jednodušší detekovat kolize mezi obrázky např. pro vývoj her. SVG je rovněž možno použít k jednoduchým animacím.

Naproti tomu Canvas API je ryze bitmapové API umožňující manipulovat s jednotlivými

pixely. Pomocí Canvas API můžeme vytvářet extrémně rychle animace. Dalo by se říci, že SVG API je v jistém smyslu *high-level* přístup a Canvas API *low-level* se vším, co to znamená.

7.3 Audio a video

7.3.1 Audio

Před příchodem HTML5 neexistovaly žádné všeobecně přijímané standardy pro přehrávání zvuku v prohlížeči. Jediným všeobecně používaným standardem pro přehrávání multimédií je Flash. Bohužel se jedná o third-party modul, se kterým např. v 64bitových distribucích Linuxu byly donedávna velké problémy z důvodu problematické podpory ze strany výrobce.

HTML5 definuje nový element `<audio>`. Audio element podporují všechny prohlížeče, ve verzích zmíněných v úvodu. Použití je přímočaré (viz ukázka 12).

Ukázka 12 Použití `audio` elementu

```
<audio controls="controls">
  <source src="music.mp3" type="audio/mpeg" />
  <source src="music.ogg" type="audio/ogg" />
  Audio not supported
</audio>
```

Atribut `controls` přidává základní ovládací prvky jako tlačítka *play*, *pause* apod. Jelikož různé prohlížeče podporují různé audio formáty, je vhodné uvést několik alternativ. Prohlížeč podle normy analyzuje jednotlivé soubory od prvního a použije první formát, kterému rozumí. Volbu MP3 akceptují všechny běžné prohlížeče kromě Firefoxu, který MP3 neobsahuje kvůli licenčním problémům. Volbu OGG naopak ignoruje např. prohlížeč Internet Explorer. Kombinace těchto dvou formátů ale pokrývá všechny uvažované prohlížeče.

7.3.2 Video

Platí analogické teze jako u elementu `<audio>`. Co se videa týče, má flash ještě mnohem dominantnější postavení. Nativní podpora přehrávání videa (a to včetně akcelerace na GPU) je ale úžasným krokem kupředu, který odstraňuje mnoho vrstev a všudepřítomných chyb ve flashových přehrávačích. Použití je analogické elementu audio a je uvedeno v úkázce 13.

Ukázka 13 Použití `video` elementu

```
<video id="vid" width="800" height="600" controls>
  <source src="movie.mp4" type="video/mpeg" />
  <source src="movie.webm" type="video/webm" />
  Video not supported
</audio>
```

7.3.3 Atributy a události elementů `audio` a `video` (JavaScript API)

Užitečné je, že se nemusíme spoléhat na nativní ovládací prvky pro přehrávání videa. Pokud neuvedeme atribut `controls`, je video zobrazeno jen samo o sobě bez čehokoliv jiného. Velice jednoduše můžeme implementovat např. požadavek, aby se video přehrávalo, právě když je nad ním umístěn kurzor myši. Uveďme si ukázku pro změnu s drobným použitím frameworku jQuery.

Ukázka 14 Pouštění videa, jen pokud je nad ním kurzor myši

```
$('#vid').hover(
function() {
  document.getElementById('vid').play();
},
function() {
  document.getElementById('vid').pause();
});
```

K dispozici máme nejrůznější atributy pro zjištění stavu přehrávání, hlasitosti atp. Atributy jsou pojmenovány vesměs velmi dobře a přehledně a jejich význam je zřejmý.

- **Společné**: chybový stav: `error`; síťové atributy: `src, currentSrc, buffered, preload, networkState`; stav přehrávání: `seeking, paused, readyState`; ovládací prvky: `controls, volume, muted`; přehrávání: `currentTime, startTime, duration, playbackRate, played, seekable, ended, autoplay, loop`.
- **Pouze video:** rozměry: `width, height, videoWidth, videoHeight, poster`.

Kromě metod `play()` a `pause()`, které jsme si ukázali v příkladu, můžeme ještě použít následující metody: `load()` – pro nahrání audia nebo videa, `canPlayType(type)` – pro dynamické zjištění, zda je daný formát v prohlížeči přehratelný a `addTrack(label, type, language)` – pro přidání další stopy do "playlistu".

7.3.4 Formát WebM

WebM je nový open-source formát původně vyvíjený společností Google speciálně pro použití v HTML5. Soubor ve formátu WebM obsahuje video v kodeku VP8 a zvuk v kodeku Vorbis. Vše je uloženo v kontejneru analogickém formátu Matroska (MKV). WebM je přímým konkurentem kodeku H.264, který není ale všeobecně užíván z analogických licenčních důvodů jako formát MP3. Ačkoliv se lze setkat s tvrzením, že formát WebM zatím za formátem H.264 zaostává co do kvality videa, rozdíly jsou minimální a vše nasvědčuje tomu, že se formát WebM stane světově používaným standardem. Formát WebM přehrává nativně Firefox, Chrome i Opera, Internet Explorer vyžaduje plugin.

8 Ostatní API

V této poslední kapitole týkající se HTML5 si uveďme některá API, která řeší poměrně unikátní problém. Vybrali jsme jednak API pro podporu asynchronního spuštění JavaScriptů na pozadí (ve zvláštním vlákně/procesu), jednak Geolocation API, jehož využití je zejména u mobilních prohlížečů.

8.1 Paralelní výpočty

Uživatelům složitějších webových aplikací se v minulosti mnohokrát stávalo, že se webová stránka začala načítat, poté jaksi na chvíli zamrzla a po pár sekundách se teprve vyrenderovala a zobrazila. K tomuto neblahému pozorování může samozřejmě dojít z mnoha důvodů, jedním z nich jsou náročnější JavaScripty, které musejí proběhnout před definitivním zobrazením stránky.

A přesně toto je smysl existence Web workers (dále jen workery). Běh JavaScriptu se provádí jednovláknově, s příchodem standardu Web workers je ale možno spustit worker na pozadí, který běží v nezávislém vlákně/procesu a se kterým je možno obousměrně komunikovat pomocí zasílání zpráv. Tím můžeme oddělit UI od náročnějších úloh, které je možno pustit na pozadí a výstup zobrazit asynchronně v callbacku. Vstupem workeru je vždy daný samostatný `*.js` soubor, který se předává v konstruktoru třídy.

```
var worker = new WebWorker('myscript.js');
worker.postMessage('Start'); // odešle zprávu workeru
```

Pro přijetí a reakci na výstup z workeru slouží událost `onmessage`.

```
worker.onmessage = function(e) {
```

```
  // e.data - libovolná data z workeru
}
```

Konečně worker je možno zrušit voláním

```
worker.terminate();
```

8.2 Geolocation

Geolocation API je určeno, jak název napovídá, pro zjištění aktuální polohy. K samotnému určení geografické pozice se používají nejlepší technologie dostupné na zařízení, na kterém je prohlížeč právě spuštěn. Využití tohoto API je zejména přínosné u mobilních zařízení, které jsou dnes běžně vybavena GPS lokátory, pomocí níž je možno určit polohu s nejvyšší přesností. Geolocation může využít následující technologie:

1. GPS – nejpřesnější, nevýhodou jsou delší časy nutné pro lokalizaci dostatečného počtu satelitů,
2. A-GPS (assitive GPS) – velmi přesné, využívá topologie buňkových mobilních sítí, pomocí triangulací určí přibližnou oblast, kde se zařízení nachází,
3. WIFI access points – mohou využít informací od poskytovatele WIFI signálu pro určení polohy,
4. IP – počítače s pevnou IP adresou mohou využít tuto adresu k přibližnému určení polohy, velmi nespolehlivé, proxy a podobné servery zcela znemožňují určení polohy.

8.2.1 Výstup

Uveďme ještě obsah objektu `coords` z algoritmu 15 v tabulce 4.

Atribut	Význam	Typ	Jednotka
`coords.latitude`	šířka	double	stupeň
`coords.longitude`	délky	double	stupeň
`coords.altitude`	výška	double/null	metr
`coords.accuracy`	přesnost	double	metr
`coords.altitudeAccuracy`	přesnost výšky	double/null	metr
`coords.heading`	směr	double/null	stupeň
`coords.speed`	rychlost	double/null	m/s
`timestamp`	časové razítko	DOMTimeStamp	–

Tabulka 4: Obsah objektu `coords`

Ukázka 15 Příklad použití Geolocation API

```
<script>
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(found, failed);
  } else {
    someDiv.innerHTML = "Geolocation is not supported.";
  }
}
function found(position) {
  someDiv.innerHTML = "Latitude: "
    + position.coords.latitude
    + "<br />Longitude: "
    + position.coords.longitude;
}
function failed() {
  alert('Oops');
}
</script>
```

8.2.2 Bezpečnost

Informace o poloze uživatele jsou citlivou informací. Proto specifikace Geolocation API vyžaduje, aby prohlížeče odesílaly informaci o poloze uživatele až po jeho explicitním odsouhlasení.

Část III

CSS3

Význam kaskádových stylů při tvorbě webových stránek či aplikací jistě nemusíme zmiňovat. V této kapitole se zaměříme pouze na nové možnosti jejich poslední verze, tedy CSS3. CSS3 obsahuje všechny funkce předchozích verzí (CSS 2.1) a přidává množství nových při zachování zpětné kompatibility. Jedná se především o vlastnosti, které umožňují vývojářům efektivně ovlivňovat vzhled stránek či aplikací bez nutnosti používat skripty. Pomocí kaskádových stylů již tedy můžeme například vytvářet oblé hrany boxů, aplikovat na objekty dvourozměrné a trojrozměrné transformace, přechody, animace apod. CSS3 je rozdělen do logických celků, tzv. modulů, které si popíšeme v následujících podkapitolách.

1 Prefixy výrobců prohlížečů

V průběhu práce na nových modulech CSS3 potřebují vývojaři určitou zpětnou vazbu od výrobců prohlížečů a také webových designérů, aby bylo možné nové CSS vlastnosti ladit. Během tohoto procesu se syntaxe a implementace jednotlivých vlastností mění. Tato skutečnost však může způsobit nepříjemnosti. Pokud totiž webdesignéři začnou používat nové vlastnosti CSS ve svých aplikacích, může se stát, že jakmile například změní určitý prohlížeč jejich implementaci, povede to k pádu zmíněných aplikací. Z těchto důvodů byly zavedeny prefixy výrobců prohlížečů (vendor prefixes), které používáme u CSS modulů nacházejících se ve stádiu vývoje. Syntaxe je uvedena v tabulce 5. Příkladem použití je např. CSS definice:

```
-webkit-border-shadow: inset 0 0 1px #ddd;
border-shadow: inset 0 0 1px #ddd;
```

Prefix výrobce	Prohlížeč
`-moz-`	Firefox
`-webkit-`	Chrome, Safari
`-ms-`	Internet Explorer
`-o-`	Opera

Tabulka 5: CSS prefixy výrobců prohlížečů

Zde vidíme použití prefixů v praxi. Jako první je uvedena vlastnost `border-shadow` s prefixem. Důvodem je, že jakmile bude ladění této funkce dokončeno, ztratí prefixy význam

a budou zrušeny.

2 Nové CSS3 selektory

Selektory známe již z předchozích verzí CSS. Slouží nám k definici, kterých elementů se bude daný styl týkat. Selektorů existuje celá řada, my se však budeme věnovat *pouze* těm z nich, které nám přinesla nově specifikace CSS3.

2.1 Selektory atributů

Slouží k výběru elementu pro styl na základě atributu a nyní i např. určité části řetězce obsaženého v hodnotě tohoto atributu.

Selektor	**Použití**
`[atribut = hodnota]`	Všechny elementy s atributem rovným dané hodnotě.
`[atribut ^= hodnota]`	Daný styl bude aplikován na všechny elementy, které obsahují uvedený atribut, jehož hodnota začíná uvedeným řetězcem.
`[atribut $= hodnota]`	Všechny elementy obsahující uvedený atribut s hodnotou končící uvedeným řetězcem.
`[atribut *= hodnota]`	Všechny elementy obsahující uvedený atribut, jehož hodnota obsahuje uvedený řetězec.

Tabulka 6: Selektory atributů

Uveďme si příklady několika praktických selektorů podle atributů:

Ukázka 16 Selektory dle atributů

```
input[type = 'text'] {  /* Input boxy */
  background: yellow;
}

/* Boxy (text-box, image-box...) téměř zprůhledníme */
div[class $= 'box'][class ~= 'faded'] {
  opacity: 0.2;
}
```

Ukázka 17 Pseudo třídy

```
:not(.box) {  /* Negace -- vše, co nemá třídu 'box' */
  color: black;
}
:empty(.box) { /* Prázdný box */ }
:enabled, :disabled { /* Povolené/zakázané */ }
:checked { /* checkbox nebo radio, aktuálně zaškrtnuté */ }
```

2.2 Pseudo třídy

Vybírají elementy podle informací, které nejsou jendoduše čitelné ze samotných názvů elementů ani jejich atributů. Tyto informace jsou spojeny spíše s aktuálním stavem stránky či aplikace a její struktury. Často chceme vybírat např. "pouze liché řádky", "elementy, které *nemají* danou třídu", nebo výběry typu "odstavce, které jsou v elementu `div` s danou třídou" atp. Právě k takovýmto složitějším výběrům slouží nové funkce CSS3. Nejlépe bude uvést vše na příkladech:

2.2.1 Pořadí synů

CSS3 umožňuje využít pořadí synů daného elementu, k dispozici je výběr `:nth-child`, `:first-child`, `:nth-last-child`[6], `:last-child` apod.

Ukázka 18 Využití pořadí synů v CSS

```
.row:nth-child(odd) {  /* Řádky, jen liché */
  background: white;
}
```

2.2.2 Kontextové výběry

Kontextové dotazy jsou rovněž velice užitečné. Způsob zápisu je velmi podobný syntaxu, který užívá všeobecně užívaný JavaScriptový framework jQuery[7].

Ukázka 19 Kontextové výběry v CSS

```
div.box > p {  /* p-potomci divů s třídou box */
  color: #ff0;
}
```

[6] Pořadí počítáno od konce.
[7] `http://jquery.com/`

3 Fonty (@font-face)

HTML5 a CSS3 zavádí nový tag `@font-face`. Pomocí něj můžeme prohlížeč instruovat, aby nahrál definici fontu z externího média (disk, síť) a podle odpovídajících CSS stylů vyrenderoval vybrané části stránky tímto fontem. Do té doby se používaly různé javascriptové knihovny, které renderovaly obrázky s použitím netradičních fontů na pozadí. Příkladem může být např. Cufón ([1]). To bylo jednak těžkopádné, ale hlavně výsledky nebyly jednotné na všech prohlížečích.

Použití `@font-face` je velice jednoduché. Nejprve v CSS font nahrajeme (triková "IE–failsafe" implementace dle [4]) a následně font standardně použijeme pomocí atributu `font-family` (viz ukázka 20).

Ukázka 20 Použití custom fontů pomocí tagu `@font-face`

```
// css
@font-face {
  font-family: 'MyBeautifulFont';
  src: url('MyBeautifulFont.eot');
  src: local(":-)"),
       url('MyBeautifulFont.otf') format('otf'),
       url('MyBeautifulFont.ttf') format('ttf');
}
.with-beautiful-font {
  font-family: 'MyBeautifulFont', serif;
}

// html
<div class="with-beautiful-font"></div>
```

Prohlížeče jsou schopny zpracovat fonty v několika formátech (více viz [3] a tabulka 7). Poznamenejme, že analogicky jako u audia a videa ani zde neexistuje formát fontu, který by byl akceptován všemi prohlížeči. Implementace použití externích fontů v CSS z [4] funguje ale ve všech prohlížečích. Dále více viz [2] nebo [5].

- OpenType (OTF), TrueType (TTF)
- Scalable Vector Graphics (SVG)
- Web Open Font Format (WOFF) – nejprogresivnější formát pro web, komprimovaný, přátelský k použití licencí fontů na webu.

- Embedded OpenType (EOT) – formát, který Microsoft prosazuje pro Internet Explorer (možná s menším úspěchem).

Prohlížeč	OTF	TTF	SVG	WOFF	EOT
Firefox	☺	☺	–	☺	–
Chrome	☺	☺	–	☺	–
Safari	☺	☺	☺	–	–
Internet Explorer	–	–	–	☺	☺
Opera	☺	☺	☺	–	–

Tabulka 7: Podpora formátů fontů v prohlížečích

4 Další nové vlastnosti textu

CSS3 přináší také řadu užitečných funkcí v oblasti formátování a stylování textu.

4.1 Text ve sloupcích

Pro sloupcovou sazbu textu máme nyní k dispozici sadu užitečných vlastností, na které se nyní zaměříme podrobněji.

4.1.1 Počet sloupců

Do jakého počtu sloupců bude text rozdělen, definujeme vlastností `column-count`. Zadáváme číslem nebo použijeme hodnotu *auto*, která nastaví počet sloupců podle ostatních omezujících parametrů, jako je například šířka sloupce.

4.1.2 Šířka sloupce

Šířku sloupce v pixelech definujeme vlastností `column-width`. Opět je možné nastavit *auto*.

4.1.3 Délka sloupců

K tomu, abychom zajistili, že sloupce budou mít vždy stejnou nebo maximálně podobnou délku, využíváme vlastnost `column-fill`. Vybíráme ze dvou hodnot: *auto* a *balance*. V prvním případě bude sazba textu do sloupců probíhat způsobem postupného vyplňování zleva doprava, takže sloupce budou pravděpodobně různě dlouhé. V druhém případě je text vysázen tak, aby délka sloupců byla stejná nebo alespoň podobná.

4.1.4 Mezera mezi sloupci

Šířku mezery mezi sloupci definujeme vlastností `column-gap`. Hodnotu zadáváme v pixelech nebo volíme *auto*.

4.1.5 Přesah textu

Často je potřeba umístit určitý text tak, aby na koncích sloupců nebyl zalomen. Jinými slovy, chceme, aby například nadpis přesahoval přes všechny sloupce. V takovémto případě využíváme vlastnost `column-span`, které nastavíme hodnotu *all*.

4.1.6 Vlastnosti předělů

Mezi sloupce je možné umístit předěly, jejichž vlastnosti nastavujeme pomocí `column-rule`. Zadáváme hodnoty v pořadí: *šířka*, *styl*, *barva*. Chceme-li definovat pouze jednu z těchto vlastností, přidáme postfix, například: `column-rule-width`.

4.2 Obecná formátovací pravidla

4.2.1 Způsob zarovnání do bloku

Způsob zarovnávání textu do bloku v případě, kdy má vlastnost (známá ze staších verzí CSS) `text-align` nastaveno *justify*. Možné hodnoty jsou: *none* (zakázáno), *auto* (řízeno prohlížečem), *inter-word* (řízeno změnami velikosti mezer mezi slovy), *inter-ideograph* (ideografický text), *inter-cluster* (pouze text, neobsahující mezery ve slovech), *distribute* (poslední řádek nezarovnávat) a *kashida* (roztahováním písmen).

4.2.2 Přetékání textu

Vlastností `text-overflow` specifikujeme, jak bude nakládáno s textem, který přeteče okraj svého nadřízeného elementu. Vybíráme z hodnot: *clip* (zkrátit text), *ellipsis* (zkrátit text a na konec řádku vložit tři tečky), *"zadaný text"* (nahradí zadaným textem).

4.2.3 Zalamování textu

Text může být zalamován několika způsoby: *normal* (pouze v místech k tomu určených), *none* (nikdy a bude-li text dlouhý, přeteče řádek), *unrestricted* (mezi kterýmikoliv dvěma znaky), *suppress* (mezi html značkami, nebude zalomen). Jednu z těchto hodnot vybereme pro vlastnost `text-wrap`.

4.2.4 Dělení slov

Můžeme povolit/zakázat rozdělování dlouhých slov za účelem zalomení řádku. Možné hodnoty vlastnosti `word-wrap` jsou: *normal* (rozdělit slova pouze na místech s definovaným bodem k rozdělení) a *break-word* (rozdělit slova v jakémkoliv místě dle potřeby).

4.3 Textové efekty

4.3.1 Obrys textu

Zde použijeme vlastnost `text-outline`. K dispozici jsou hodnoty: *thickness* (zesílení obrysů), *blur* (rozostření obrysů), *color* (barva obrysů).

4.3.2 Stín vržený textem

Pro tento efekt použijeme vlastnost `text-shadow`, které nastavíme čtyři hodnoty: horizontální a vertikální posun vůči textu, míru rozzostření a barvu. První tři hodnoty zadáváme v pixelech.

5 Transformace

Další z novinek CSS3 je sada metod, díky kterým můžeme měnit geometrické vlastnosti a polohu objektů. Používáme vlastnost *transform* a jako hodnotu uvádíme jednu z nových metod.

5.1 2D transformace

5.1.1 Změna velikosti

Pro změnu velikosti elementu používáme metodu `scale()`. Jako vstupní parametry předáváme novou šířku a výšku našeho elementu v násobcích původních hodnot (1.0 je tedy beze změny). Zadáme-li pouze jednu hodnotu, změní se celá velikost se zachováním poměru stran. Ukázka následuje.

Ukázka 21 Stín vržený textem

```
// css
h1 {
        /* stín za nadpisem h1 */
        text-shadow: 8px 10px 4px blue;
        /* nadpis bude umístěn přes všechny sloupce */
        column-span: all;
}

#preteceni {
        width: 200px;
        height: 25px;
        overflow: hidden;
        white-space: nowrap;
        /* text ukončen třemi tečkami */
        text-overflow: ellipsis;
        background-color: lightBlue;
}

#sloupce {
        column-width: 150px; /* šířka sloupce */
        /* dvojité předěly mezi sloupci*/
        column-rule-style: double;
        column-rule-color: lightBlue; /* barva předělů */
}
```

Ukázka 22 Změna velikosti

```
// css
#vetsi {
        /* objekt bude mít 120% puvodní šířky
           a 150% původní výšky */
        transform: scale(1.2, 1.5);
}
// html
<div id="vetsi"> ... </div>
```

5.1.2 Změna pozice

Pomocí metody `translate()` můžeme posouvat objektem o stanovený počet pixelů. První parametr této metody udává horizontální posun zleva, druhý pak vertikální posun shora.

Ukázka 23 Změna pozice

```
// css
#posuv {
        transform: translate(20px, -10px);
}
// html
<div id="posuv"> ... </div>
```

5.1.3 Rotace

Otáčení elementem po směru hodinových ručiček realizujeme metodou `rotate()`. Zadáváme počet stupňů, o které bude element otočen.

Ukázka 24 Rotace

```
// css
#rotace {
        transform: rotate(45deg);
}
// html
<div id="rotace"> ... </div>
```

5.1.4 Zkosení

Metoda `skew()` slouží ke zkosení elementu ve dvou osách. Jako parametry předáváme úhel, který bude svírán mezi osou y a vertikální hranou objektu, a dále úhel mezi osou x a vertikální hranou objektu.

Ukázka 25 Zkosení

```
// css
#zkoseni {
        transform: skew(15deg, 30deg);
}
// html
<div id="zkoseni"> ... </div>
```

5.1.5 Kombinace transformací

Můžeme také definovat všechny výše uvedené transformace v jedné metodě `matrix()`. Parametry vkládáme v tomto pořadí: *šířka*, *horizontální zkosení*, *vertikální zkosení*, *výška*,

horizontální posun, *vertikální posun*.[8]

Ukázka 26 Obecná lineární transformace s posunem

```
// css
#matrix {
        transform: matrix(0.2,0.5,0.5,0.2,10,10);
}
// html
<div id="matrix"> ... </div>
```

Metody *translate()*, *scale()* a *skew()* mají ještě své alternativy, které použijeme pro případ transformace pouze v jedné ose. Název metody je pak tvořen přidáním postfixu X, nebo Y podle směru transformace, kterou vyžadujeme. Taková metoda pak přirozeně přijímá pouze jeden parametr.

5.2 3D transformace

3D transformace využívají stejné metody jako 2D transformace s výjimkou metody pro zkosení `skew()`, která není v trojrozměrném prostoru použitelná. Máme tedy k dispozici metody: `scale3d()`, `rotate3d()`, `translate3d()` a `matrix3d()`. U všech těchto metod přibývá parametr udávající transformaci v ose *z*. Opět můžeme využít také metody pro transformaci v jedné z os, například `scaleZ()`.

5.3 Vlastnosti pro další nastavení transformací

Kromě vlastnosti `transform` a transformačních metod využíváme také dalších vlastností, kterými nastavujeme, jak bude výsledná transformace vypadat.

5.3.1 Poloha základny pro trasformaci

Definujeme posun bodu v nadřízeném elementu vlastností `transform-origin`. Podle tohoto bodu bude daná transformace provedena. Vybíráme z hodnot *left*, *right*, *center* nebo zadáme posun základny v procentech šířky a výšky nadřízeného elementu.

[8] Řečeno jazykem lineární algebry, matice 2×2 nám definuje libovolnou 2D lineární transformaci, ke které přidáme ještě obecný posun, který lineární transformací není.

5.3.2 3D vykreslování

Cheme-li elementy vykreslovat jeko 3D objekty, nastavíme vlastnost `transform-style` na hodnotu *preserve-3d*. V opačném případě zvolíme hodnotu *flat*.

5.3.3 Perspektiva

Perspektivu 3D vykreslení vnořeného objektu definujeme pomocí vlastnosti `perspective`. Jedná se vlastně o posouvání začátku osy z od pomyslné plochy prohlížeče (monitoru). Hodnotu zadáváme v pixelech.

5.3.4 Poloha základny pro 3D trasformaci

Vlastnost `perspective-origin` má stejný význam jako výše uvedená vlastnost `transform-origin`, zde ovšem pro trojrozměrnou transformaci.

5.3.5 Viditelnost 3D objektů

Při otáčení trojrozměrným objektem můžeme vyžadovat, aby nebyl viditelný, jakmile k nám bude otočen svou zadní stranou. Toho docílíme nastavením vlastnosti `backface-visibility` na hodnotu *hidden*. V opačném případě volíme *visible*.

6 Animace

Animace v CSS3 jsou podobné přeměnám (*transitions*). Rozdíly jsou především ve způsobu spouštění a složitosti. Zatímco transition je spouštěna reakcí na určitou změnu vlastností CSS, například najetí myší na element nebo pomocí JavaScriptu, animace mohou být spouštěny ihned po načtení stránky prohlížečem bez nutnosti jakékoliv interakce ze strany uživatele. V animacích navíc používáme pomocné body, tzv. *keyframes*, které nám umožní úplnou kontrolu při jemnějším nastavování průběhu animace. Objekt, který chceme animovat, musí mít ve svém stylu vlastnost *animation*, jejíž hodnota odpovídá názvu animace popsané sestavou stylů *@keyframes*.

6.1 Vlastnosti pro nastavení animací

6.1.1 Délka trvání jednoho cyklu animace

Délku trvání jednoho cyklu animace nastavujeme v sekundách pomocí vlastnosti `animation-duration`.

6.1.2 Zpoždění

Stejně jako u přechodů i zde máme možnost nastavit čas, který uběhne, než bude animace spuštěna. Použijeme k tomu vlastnost `animation-delay`.

6.1.3 Časování

Časovou funkci volíme podobně jako v případě přechodů, zde ovšem použijeme vlastnost `animation-timing-function`.

6.1.4 Počet cyklů

Ve výchozím nastavení proběhne naše animace pouze jednou. Chceme-li vynutit běh animace vícekrát, potřebujeme k tomu vlastnost `animation-iteration-count`. Hodnotou může být číslo reprezentující počet cyklů animace. Nekonečný běh jednoho cyklu za druhým zajistíme hodnotou *infinite*.

6.1.5 Obrácený běh

Pokud animace beží nepřetržitě, vždy po skončení jednoho cyklu je spuštěn další od začátku. Můžeme ovšem chtít, aby každý další spuštěný cyklus běžel vždy obráceně než předchozí. Animace pak beží střídavě dopředu a pozpátku. Toho docílíme definováním vlastnosti `animation-direction`. Možné hodnoty jsou *normal* a *alternate*.

6.1.6 Identifikátor animace

Styl objektu, který bude animován, musí obsahovat vlastnost `animation-name`, jejíž hodnota je shodná s názvem sady stylů `@keyframes`, kde je celá animace popsána.

6.1.7 Pozastavení/spuštění

Průběh animace lze pozastavit a opět spustit nastavením hodnoty vlastnosti `animation-play-state` na *pause* nebo *run*.

6.1.8 Kumulovaná vlastnost

Vlastnost `animation` slouží k nastavení všech výše zmíněných vlastností do jedné. Samozřejmě s výjimkou `animation-play-state`. Hodnoty nastavujeme v tomto pořadí: *name*, *duration*, *timing-function*, *delay*, *iteration-count*, *direction*.

6.2 Nastavení pravidel a stylů pro animaci

Nyní se dostáváme k samotnému popisu naší animace. Jedná se o seznam definicí stylů, které budou na daný element postupně během animace aplikovány. Před každý z těchto stylů uvádíme procentuální hodnotu časového bodu, ve kterém se animace musí nacházet, aby element plně přešel do definovaného stylu. Tento seznam je označen klíčovým slovem `@keyframes`, za nímž následuje název (identifikátor) animace. Vše vidíme v ukázce 27.

Ukázka 27 CSS3 animace

```
// css
#animace { /* počátečí styl */
        width:100px;
        height:100px;
        background:lightBlue;
        /* bude použita animace a bude trvat 5 sekund */
        animation:animace 5s;
}

@keyframes animace {
/* objekt postupně opíše trojúhelník, otočí se
        a změní barvu */
        0%   {transform:translate(0, 0);}
        20%  {transform:translate(200, 0);}
/* v 40% běhu animace bude element posunut o 100px zleva
        a 200px zprava */
        40%  {transform:translate(100, 200);}
        100% {transform:rotate(180deg);
                background-color:blue;}
}
// html
<div id="animace"> ... </div>
```

Pozn.: Místo 0% a 100% je možné použít hodnoty *from* a *to*.

7 Okraje a pozadí elementů

Zde máme k dispozici nové užitečné nástroje, pomocí kterých jednoduše definujeme vzhled našich elementů. Jako nejpřínosnější v této kategorii jsou bezesporu vlastnosti pro zaoblení elementů a vícenásobná pozadí.

7.1 Pozadí

7.1.1 Velikost obrázku pozadí

Velikost obrázku na pozadí definujeme jeho šířkou a výškou v pixelech či procentech jeho nadřízeného elementu. Můžeme také využít automatické nastavení velikosti, a to ve dvou režimech: *contain* a *cover*. V prvním případě bude obrázek roztažem tak, aby se dotýkal bližších protilehlých okrajů nadřízeného elementu. V druhém případě se bude dotýkat vzdálenějších okrajů.

Ukázka 28 Velikost obrázku na pozadí

```
// css
.background {
        background: url(obr1.gif)
        background-size: cover;
}
// html
<div class="background"> ... </div>
```

7.1.2 Pozice obrázku pozadí

Způsob, jakým bude pozadí zobrazováno, nastavujeme výběrem: *padding-box*, *border-box* a *content-box*. První z nich volíme v případě, kdy má být pozadí zarovnáno k vnitřní hraně rámečku nadřízeného elementu. Druhý zarovnává obrázek až k vnější hraně rámečku, poslední *content-box* zarovnává k obsahu daného elementu, jak je vidět v příkladu.

Ukázka 29 Pozice obrázku na pozadí

```
// css
.background {
        background: url(obr1.gif);
        background-origin: content-box;
        background-repeat: no-repeat;
}
// html
<div class="background"> ... </div>
```

7.1.3 Rozsah oblasti s pozadím

Rozsah oblasti s pozadím můžeme definovat pomocí vlastnosti `background-clip`. Využíváme k tomu stejných hodnot jako v předchozím případě, zde ovšem ovlivňujeme, které

části daného elementu budou vyplněny oblastí s pozadím.

Ukázka 30 Rozsah oblasti s pozadím

```
// css
#background {
        background-color: lightBlue;
        background-clip: content-box;
}
// html
<div class="background"> ... </div>
```

7.1.4 Více obrázků na pozadí

Velice užitečná je také možnost přidat objektu více než jen jeden obrázek na pozadí. Obrázky jsou vrstveny jeden na druhý a ve spojení s nastavením polohy v ose *z*, průhledností, transformacemi či přechody, můžeme snadno vytvořit graficky vytříbené elementy. V CSS stačí jako hodnotu vlastnosti background uvést seznam obrázků. Tento seznam je pochopitelně dynamicky generovatelný pomocí JavaScriptu, takže můžeme jednoduše implementovat například efektně vypadající fotografickou slideshow bez nutnosti použít nástroje třetích stran.

Ukázka 31 Více obrázků na pozadí

```
// css
#background {
/* na pozadí objektu budou použity dva obrázky najednou */
        background: url(obr1.png), url(obr2.png);
}
// html
<div class="background"> ... </div>
```

7.2 Okraje elementů

7.2.1 Zaoblení

Jedná se o velice užitečnou CSS3 vlastnost, která je pro webdesignéry skutečným přínosem, vzpomeneme-li, jakými způsoby bylo nutné tento efekt provádět dříve. Nyní stačí nastavit požadovaný rádius v pixelech jako hodnotu atributu `border-radius`.

7.2.2 Stín

Náš element může také vrhat stín, což zajistíme pomocí atributu `box-shadow`. Zadáváme čtyři hodnoty, přičemž první dvě definují x a y souřadnice posunu vůči elementu, další pak míru rozmazání stínu a poslední nastavuje barvu.

7.2.3 Okraj z obrázku

Využíváme vlastnost `border-image`, která nám umožňuje vytvoření okrajů elementu z obrázku. Definujeme pět hodnot: cestu k souboru obrázku, horizontální posun, vertikální posun a výběr, zda má být obrázek roztažen na celou šířku a výšku daného okraje, či zda má být v okraji opakován (*stretch*, *round*).

7.2.4 Příklad

Ukázka 32 Okraj z obrázků

```
// css
div { /* společné vlastnosti */
        background: lightBlue;
        width: 100px;
        height: 40px;
}
#obleRohy {
        border-radius: 25px;
}
#stin {
        box-shadow: 20px 10px 10px grey;
}
#obrOkraj {
        border-image: url(obr_okraj.png) 40 20 round;
}
```

Část IV

ASP.NET (MVC3, Razor)

Paralelně existující předchůdce frameworku MVC3 je ASP.NET Web Forms. Začněme tuto kapitolu bodovým přehledem největších výhod, které MVC3 oproti Web Forms skýtá:

- Programátor má plnou kontrolu nad generovaným HTML kódem tak, jako tomu je u skriptovacích jazyků jako je např. Python,
- MVC3 framework programátora vede *při správném použití* k separaci zájmů (separation of concerns – SoC; oddělení různých vrstev aplikace v maximální možné míře). Tento bod je velmi důležitý, neboť není složité psát controllery špatně,
- Absence `ViewState` a `PostBack` událostí, díky kterým byl vývoj ve Web Forms poměrně značně nepřehledný,
- Perzistentní URL, které umožňují bookmarkování a SEO,
- Naprosto přímočará integrace s JavaScript frameworky – nutnost pro použití HTML5,
- Umožňuje Test Driven Development (TDD) – psaní testů a jejich automatické spouštění,
- Razor – nový velmi efektivní templatovací jazyk.

1 MVC (Model–View–Controller) paradigma

Model–View–Controller je návrhový vzor, podle kterého je aplikace rozdělená na tři hlavní části: *model*, *view* a *controller*. Microsoft ASP.NET MVC framework představuje alternativu k frameworku ASP.NET Web Forms, oba jsou určeny pro vývoj webových aplikací. Aktuální návrh obou frameworků umožňuje oba tyto přístupy v rámci jedné aplikace libovolně kombinovat, přestože je otázkou, zda toto povede k vyšší přehlednosti aplikace.

1.1 Model

Model implementuje nejnižší vrstvu logiky aplikace na úrovni dat. Zejména se jedná o činnosti jako získání a aktualizace dat databáze, základní transformace dat (např. do objektové podoby). Model je *zcela nezávislý* na view a controllerech.

1.2 View

View zobrazuje (poupravená, přepočítaná) data modelu. Dále je view zodpovědné za generování uživatelských akcí (např. kliknutí na tlačítka) zpět na controllery.

1.3 Controller

Controller je součástí aplikace, která akceptuje požadavky uživatele, provádí výpočty a rozhoduje o tom, jaké *view* se bude renderovat. Těmto view také hlavně poskytuje data získaná z modelu. Controller závisí jednak na view, jednak na modelu. Controllery jsou hlavní výkonnou složkou MVC návrhu.

✎ Model reprezentuje data. Nezávisí nikdy na controlleru ani view.

2 Nový templatovací jazyk Razor

Razor je nový templatovací view engine pro ASP.NET. ASP.NET ve formě MVC vždy podporovalo templatování, nejpoužívanější jsou ASPX/ASCX šablony. Razor ale přichází s mnoha vylepšeními. Základní principy tohoto jazyka jsou:

- *Kompaktnost* – Razor rapidně snižuje počet tagů a znaků nutných pro zápis šablon. Oproti jiným šablonovacím jazykům má velmi inteligentní detekci výkonného (v tomto případě C#) kódu v šablonách.
- *Známost* – Razor striktně využívá C# jako svůj výkonný jazyk, není se tedy nutno učit téměř nic nového.
- *Žádné nástroje* – Razor nepotřebuje žádné nástroje (tools) k vytváření šablon. Vše, co je potřeba, je jakýkoliv textový editor.
- *Testovatelnost* – Šablony v Razoru lze dle moderních přístupů k programování automaticky testovat.
- *Nezávislost* – Razor je možno použít k templatování i zcela mimo ASP.NET v jiných typech aplikací. Toto považujeme za výborný a velmi přirozený fakt.

Nyní si ukažme základní vlastnosti Razoru na příkladech. Kde to bude vhodné, budeme srovnávat ASPX s Razorem.

2.1 Syntax Razoru

Ukážeme si, jak zapsat v razoru základní konstrukce:

- Odkazy na proměnné
- Cykly
- Podmínky

Proměnné

Ukázka 33 Použití proměnných v ASPX

```
<h1>ASPX style</h1>
Your name is <%=username %>.
Current date is <%= DateTime.Now %>.
Detail is <a href="/Detail/<%=userId %>">here</a>.
```

V Razoru je tento snippet o něco přirozenější a přehlednější. Odkazy na proměnné jsou uvozeny znakem `@`.

Ukázka 34 Použití proměnných v Razoru

```
<h1>Razor style</h1>
Your name is @username.
Current date is @DateTime.Now.
Detail is <a href="/Detail/@userId">here</a>.
```

Parser v Razoru je velice chytrý a dokáže identifikovat odkazy a použití výkonného kódu (ne-HTML) v šabloně bez nutnosti kód nějak velmi speciálně uvozovat.

✎ Na proměnné se odkazujeme v Razoru jako `@variable`.

Cykly

V situacích, kdy chceme vyrenderovat všechny objekty daného seznamu, např. jako list `<ul>` nebo i jako tabulku `<table>`, je nutno použít cyklus a v těle cyklu definovat, jak vyrenderovat jeden prvek nebo řádek. Pomocí ASPX tuto standardní úlohu řešíme takto:

Ukázka 35 Použití cyklů v ASPX

```
<h1>ASPX cycles</h1>
<ul>
  <% foreach (var i in people) { %>
    <li><%= i.name %>, <%= i.age %></li>
  <% } %>
</ul>
```

V příkladu vidíme, že symbolů procento začíná přibývat a kód se tak stává méně přehledným. Každou výkonnou sekci je také nutno explicitně uzavřít (předposlední řádek generující pouze uzavírající závorku těla cyklu). Naproti tomu v Razoru je tento jednoduchý kód o mnoho elegantnější.

Ukázka 36 Použití cyklů v Razoru

```
<h1>Razor cycles</h1>
<ul>
  @foreach (var i in people) {
    <li>@i.name, @i.age</li>
  }
</ul>
```

Tento kód demonstruje, jak velmi jednoduchý a úsporný je jazyk Razor. Těžko si lze představit, že bychom tuto úlohu vygenerování všech odrážek vyřešili jednodušeji a úsporněji.

✎ Výkonný kód šablony není potřeba kromě znaku `@` nijak speciálně uvozovat a ani končit. Parser Razoru chytře rozpozná HTML kód od výkonného C# kódu.

Podmínky

Podmínky lze rovněž zapsat velice přirozeně. Chceme-li např. vyrenderovat řetězce `"nezáporné"` pro kladnou hodnotu parametru a `"záporné"` v opačném případě, docílíme toho takto:

Ukázka 37 Použití podmínky v Razoru

```
<h1>Razor simple if</h1>
@if(x >= 0) {
  <span>nezáporné</span>
} else {
  <span>záporné</span>
}
```

Víceřádkové a vícetokenové výrazy

Ukažme si ještě, jak lze v Razoru vložit víceřádkový výkonný kód a jak lze vložit výkonný kód přes více tokenů. Obojí je opět velice jednoduché. Víceřádkové výrazy se uvozují `@{` a zavírají normální složenou závorkou. Vícetokenové výrazy se uvozují `@(` a uzavírají kulatou závorkou.

Ukázka 38 Víceřádkové a vícetokenové výrazy v Razoru

```
<h1>Razor multi-line/token</h1>
@{
  int x = 1;
  string msg = "Computed value is " + x + ".";
}
<p>Result: @msg</p>
<p>Result: @("Computed valus is " + x + ".")</p>
```

2.2 Layout

Při tvorbě webové aplikace typicky udržujeme jednotnou strukturu stránek. Nahoře bývá menu a hlavička, na straně toolbox nebo kontextové menu, uprostřed obsah a dole nějaká patička se souhrnem linků nebo s doplňkovými informacemi o copyrightu a podobně. ASP.NET ve verzi 2.0 užívalo tzv. *master pages*, díky kterým jsme mohli definovat základní layout a pak tzv. *detail pages*, ve kterých byly definovány konkrétní stránky aplikace (např. seznam uživatelů, výpis produktů atp.). Razor zachovává základní ideu, ale umožňuje mnohem pružněji využívat dědičnosti šablon apod. Inspiruje se přitom u analogického fungování takovýchto šablon v jiných jazycích. Autor má zkušenosti s templatovacím jazykem Mako[9] pro Python, ve kterém tyto principy fungují již delší dobu.

Naši ukázku layoutování začněme s controllerem `SimpleController` a jeho jedinou metodou `Index`. Tento controller udělá jen to, že nahraje z testovací databáze Northwind

[9] `http://makotemplates.org/`

Ukázka 40 Příklad Razor šablony

```
@model IList<Sample.Models.Category>

<!doctype html>
<html>
<head>
</head>
<body>
  <h1>List of Categories</h1>
  <ul>
    @foreach(var c in Model) {
      <li>@c.CategoryName</li>
    }
  </ul>
</body>
</html>
```

seznam kategorií a pošle je do šablony.

Ukázka 39 Obecný controller

```
public class SimpleController : Controller {
  NorthwindDB database = new NorthwindDB();

  public ActionResult Index() {
    var list = database.Categories.ToList();
    return View(list);
  }
}
```

Nyní si ukažme odpovídající soubor se šablonou `Index.cshtml`, který ale ještě nebude využívat výhody layoutování v Razoru.

Kdybychom takto psali celou webovou aplikaci, stále bychom duplikovali HTML hlavičky, nahrávání CSS a JavaScriptů na všech stránkách a výsledkem by byl velice těžko udržovatelný kód, který by se bortil při každé drobné změně. Porušovali bychom základní princip DRY.

Kostra správného řešení ve stylu MVC3 vypadá následovně. Nejprve definujeme unikátní soubor `_Layout.cshtml` (ukázka 41).

Ukázka je velice jasná, až na volání `@RenderBody()`. Toto volání je právě klíčem k použití šablon v MVC3, neboť označuje místo, kam se vloží obsah, který je výstupem z

Ukázka 41 `_Layout.cshtml`

```
<!DOCTYPE html>
<html>
  <head>
    <title>@Page.Title</title>
    <link href="~/Styles/Main.css" rel="stylesheet"
                type="text/css" />
  </head>
  <body>
    <div id="container">
      <div id="header">
        <h1>Web Application</h1>
      </div>
      <div id="main">@RenderBody()</div>
      <div id="footer">&copy; @DateTime.Now.Year
                Michal Danek</div>
    </div>
  </body>
</html>
```

konkrétního volání (renderování) view. Funguje to tedy následovně.

1. Vytvoříme soubor `_Layout.cshtml` s základním layoutem celé aplikace a hlavně se všemi potřebnými odkazy na skripty a kaskádové styly.

2. Vytvoříme konkrétní šablony, ve kterých příkazem `Layout = ~/_Layout.cshtml;` nastavíme, že se má daná šablona vyrenderovat v rámci daného layoutu.

Část V

Ukázková implementace – HTML5 Web Demo

Jako ukázku technologií HTML5 jsem připravil vlastní webovou aplikaci, která má za úkol představit ty novinky z HTML5, které podstatnějším způsobem zvyšují uživatelský komfort. Webová aplikace je úložiště tzv. *stránek (pages)*, kdy každou jednu stránku můžeme chápat jako nástěnku, na kterou si uživatel "natahá" obsah. Stránky je pak možno pojmenovávat, otagovat a hlavně prolinkovat mezi sebou, čímž dostaneme klasicky WIKI systém, ale bez nutnosti psát jakýkoliv zdrojový kód.

Na každou stránku můžeme umisťovat obdélníkové boxy níže uvedených typů. Boxy je možno:

- *Umístit libovolně* (zarovnáváme na grid 16x16)
- Zvětšovat a zmenšovat
- *Vnořovat* do sebe. Přetahování otcovského boxu pak přemísťuje i jeho potomky.

Můžeme použít aktuálně tyto typy boxů:

1. *Textový box* – libovolný textový obsah. Editovatelné in-place díky novým HTML5 atributům. Je možno nastavovat speciální styl pro části textu. Stačí označit část textu a stisknout kombinaci `Ctrl+Alt-1`. Objeví se následně dialogové okno a kliknutím na tlačítko je nastaven pro tento výběr nový styl. Stylování je možno z výběru také odebrat.

2. *Hyperlinkový box* – odkazy a to dvou typů: *interní* a *externí*. Jednotlivé odkazy se píšou samostatně na řádky. Interní odkazy jsou odkazy na jméno stránky v rámci WIKI. Příkladem interního odkazu je `@testpage` (předpokládáme, že existuje stránka se jménem `testpage`). Externí odkazy jsou pak libovolná URL, např. `http://www.google.com/`. Odkazy se přitom otevírají vždy v nové záložce.

3. *Obrázkový box* – demonstruje mnoho novinek z HTML5. Cílem boxu je zobrazovat slideshow obrázků, které se cyklicky prolínají. Nejprve vytvoříme dostatečně veliký obrázkový box a poté do něj metodou drag & drop myší natáhneme jednotlivé obrázky a to odkudkoli, například z Průzkumníka. Přetáhnout můžeme i několik obrázků

najednou. Obrázky jsou poté přes `FileReader API` načteny a po 1 MB velikých úsecích uploadovány na server. Tam jsou uloženy a následně na pozadí zobrazovány v boxu. K zobrazování jsou využity pokročilé metody CSS3 (transitions).

4. *Tabulkový box* – podporujeme i strukturovaný obsah. Tabulkový box můžeme použít tak, že v Excelu vytvoříme tabulku, tu označíme a celou ji přetáhneme do prohlížeče. Aplikace následně detekuje strukturovaný obsah a vytvoří speciální typ boxu – tabulku. Tabulky je možno následně editovat s tím, že je nutno zachovat interní tab-delimited formát. Tabulkové boxy (matice) řádu $N\times 2$ je možno použít pro poslední typ boxů a to jsou sloupcové grafy.

5. *Grafový box* – sloupcové grafy. Vstupem tohoto boxu je vždy nějaký tabulkový box (ten je možno vytvořit přetažením z Excelu). Grafy se refreshují vždy po reloadu stránky. Pokud tedy změníme obsah podkladové tabulky a chceme překreslit graf, je nutno nejprve stránku uložit a následně refreshovat. Pak dojde k překreslení grafu. Ke kreslení grafů využíváme nové SVG API HTML5.

V aplikaci je rovněž přístupná HTML podoba této bakalářské práce, a to v poslední záložce menu *Bakalářská práce*. WIKI obsahuje systém autentikace a správy uživatelů, obsah je ale kolaborativní, jako to bývá u WIKI zvykem. Systém uživatelských práv řešen není, nebyl předmětem práce.

Datová vrstva

Obsah je ukládán ve dvou úložištích:

- Souborový systém – obrázky jsou ukládány v adresářové struktuře přímo na souborovém systému. To je pro binární data, u kterých nezkoumáme vnitřní strukturu optimální.

- Databáze MS SQL Server – veškerá ostatní data, jako jsou definice stránek nebo informace o uživatelích, jsou ukládána databázi. Pro přístup do databáze používáme objektově-relační Entity Framework (aktuálně ve verzi 4). Datový model je možno vidět v souboru `webGuiModel.edmx`. Pro serializace a deserializaci jednotlivých WIKI stránek používáme formát JSON, který konstruujeme a parsujeme rekurzivní funkcí (boxy je možno libovolně vnořovat).

Část VI

Závěr

V této práci jsem se zaměřil na přehledový popis všech nejdůležitějších novinek v HTML5 a CSS3, které je možno využít při tvorbě interaktivnějších a hlavně uživatelsky příjemnějších aplikací. Z mého pohledu nejzajímavějšími přínosy HTML5 na klientské straně jsou nativní podpora drag & drop, canvas nebo použití lokálních úložišť a přímého přístupu k souborovému systému. Užitečná je konečně i nativní podpora zvuku a videa, kdy zejména mobilní prohlížeče mají obrovské problémy s dosud používaným flashem. Každá *third-party* závislost, které se zbavíme, je určitě krok správným směrem.

HTML5 ale nezahrnuje jen úpravy klientské strany resp. lépe řečeno vizuální složky. Mezi nejzajímavější další (spíše low-level) přínosy patří standard WebSockets pro efektivní obousměrné asynchronní zasílání zpráv nebo WebWorkers pro podporu vícevláknových JavaScriptových výpočtů na pozadí s asynchronním získáním výstupů.

Dospěl jsem k závěru, že HTML5 a CSS3 představují pro web a webové aplikace zásadní kroky vpřed. Web se díky těmto mnoha vylepšením přibližuje svou vyjadřovací silou k desktopovým aplikacím, ale ponechává si stále svou lehkost, efektivnost a multiplatformost. Dnes striktní oddělení obsahu (HTML) a vzhledu (CSS) je maximálně flexibilní a umožňuje velice efektivní práci se vzhledem aplikací.

Fenoménem dneška jsou mobilní aplikace a zařízení (mobily, tablety...). A právě posílení HTML standardu umožňuje přenést většinu kompetencí a povinností vývoje na stranu fundovaných výrobců prohlížečů (Mozilla, Google...) a umožnit tak developerům soustředit se více na svůj vlastní vývoj a jednodušeji vytvářet velice funkční aplikace pro mobilní zařízení. Tyto aplikace pak díky standardu HTML5 mohou využívat i jinak interně složité nativní funkce jako např. GPS.

Obrázek 1: Náhled stránek kreslením na canvas

Obrázek 2: Příklad WIKI stránky s několika druhy boxů

Reference

[1] Cufon – fonts for the people. `http://cufon.shoqolate.com/generate/`.

[2] E. Weyl A. Goldstein, L. Lazaris. HTML5 *and* CSS3 *for the Real World*. Sitepoint, 2011. ISBN 978-098-0846-904.

[3] ArtzStudio. 2012. `http://www.artzstudio.com /2012/02/web-font-performance-weighing-fontface-options-and-alternatives/`.

[4] Paul Irish. Bulletproof smiley @font-face syntax. 2010.

[5] M. MacDonald. HTML5 – *the missing manual*. O'Reilly, 2011. ISBN 978-144-9302-399.

[6] StatCounter. Globalstats. 2012. `http://gs.statcounter.com/`.

[7] W3C. Html canvas 2d context. 2012. `http://dev.w3.org/html5/2dcontext/`.

[8] Nicholas C. Zakas. *Professional JavaScript for Web Developers*. John Wiley & Sons, Inc., 2012. ISBN 07-645-7908-8.

Přílohy

Odkazy na dema HTML5 a CSS3

1. `http://www.giantflyingsaucer.com/blog/?p=2250` – TTF font
2. `http://www.fontsquirrel.com/fontface` – Několik set fontů, organizovaných do kategorií
3. `http://www.google.com/webfonts` – Google free fonty
4. `http://css3.bradshawenterprises.com/` – tutoriál
5. `http://www.apple.com/html5/` – Pár Apple ukázek, pěkné galerie a ukázka možností tvarování fontů (průhlednosti, stíny, rotace...)
6. `http://www.rgraph.net/` – grafy RGraph v HTML5 Canvas
7. `http://tutorialzine.com/2011/09/html5-file-upload-jquery-php/` – Drag&Drop obrázků, včetně bargrafu a vícenásobného uploadu
8. `http://net.tutsplus.com/tutorials/html-css-techniques/25-html5-features-tips-and-techniques-you-must-know/` – 28 věcí z HTML5, které musíme znát :-)
9. `http://slides.html5rocks.com/#landing-slide` – Pěkná prezentace nových HTML5 vlastností
10. `http://www.codeproject.com/Articles/237411/Html-5-Controls-for-ASP-Net-MVC` – spolupráce HTML5 a ASP.NET
11. `http://html5demos.com/` – HTML5 dema a příklady
12. `http://webdesignerwall.com/trends/47-amazing-css3-animation-demos` – 47 úžasných animací využívajících CSS3
13. `http://prohlizece.info/bakalarska-prace/index.html#css` – bc práce o CSS3
14. `http://css3generator.com/` – CSS3 generátor
15. `http://demo.tutorialzine.com/2010/06/css3-minimalistic-navigation-menu/demo.html` – Demo moc pěkného menu CSS3

16. `http://css3menu.com/elegant-dark-menu.html` – Pěkné menu CSS3
17. `http://css3generator.com/` – CSS3 generátor
18. `http://www.html5canvastutorials.com/` – Canvas tutoriál
19. `http://tutorials.jenkov.com/svg/index.html` – SVG tutoriál
20. `https://developers.google.com/chart/image/docs/gallery/line_charts?hl=cs` – Lineární grafy
21. `http://www.europe.match.com/` – Komponenta na střídání obrázků